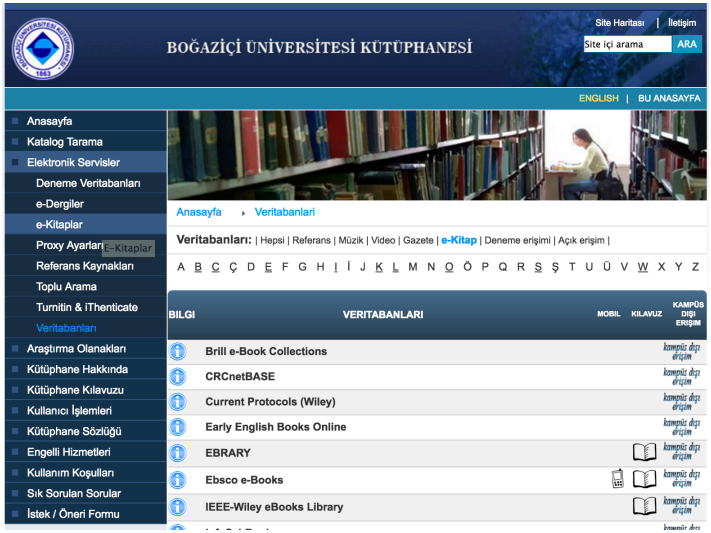
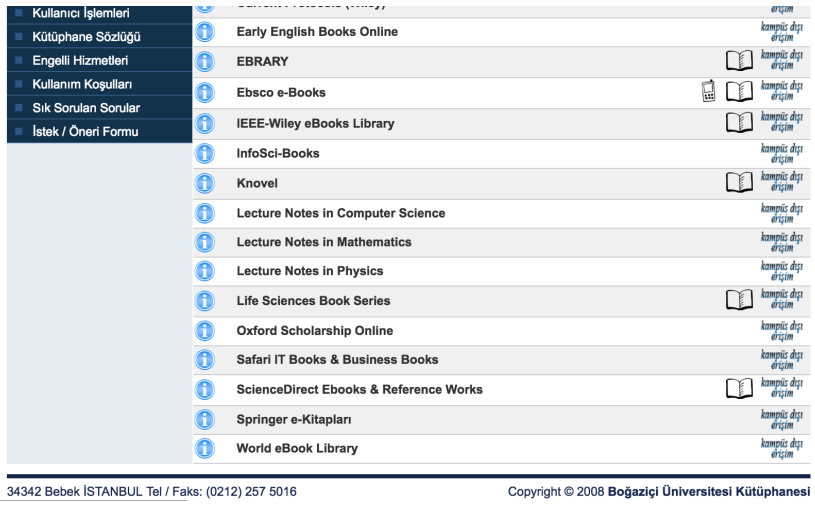
# QT Programming

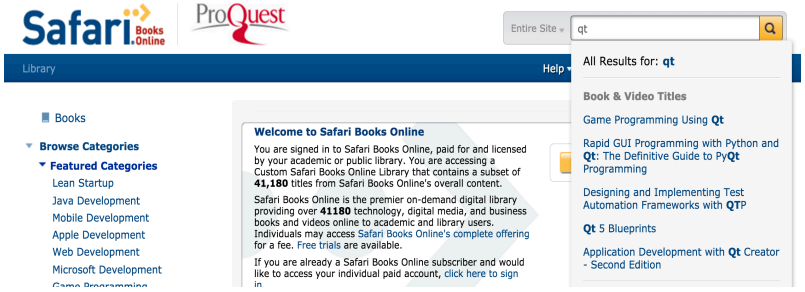
# QT Book in the Library



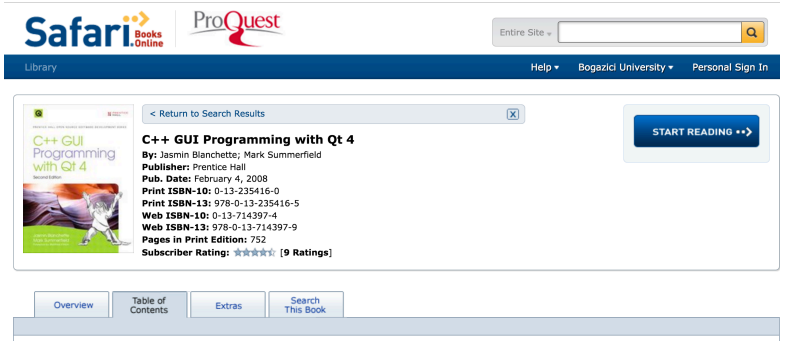

E-kitaplar (e-books)




Safari IT Books




Search qt




Read: C++ GUI Programming with Qt-4

# GUI Programming

- GUI programs use event driven programming model:
  - GUI programs have a lot of interactivity: there are many inputs that may come in different forms (mouse clicks, button clicks, drawing etc. )
  - A style of coding where a program's overall flow of execution is dictated by events.
  - The program loads, creates visual/non-visual objects and then waits for user input events.
  - As each event occurs, the program runs particular code to respond.

- Control flows of GUI programs have the following pattern

set-up, creation, configuration of objects (visual or non-visual)

```
do {
      event = GetNextEvent() ;
      process event
}  while (event != quit);
```

# HELLO QT Program


```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
     int rc ;
     QApplication app(argc, argv);
     QLabel label("Hello Qt!");

     label.show();

     rc = app.exec();
     return(rc) ;
}
```

# QApplication Object

- The QApplication class manages the GUI application's control flow and main settings.
- QApplication specializes GUI application with some functionality needed for widget-based applications. It handles widget specific initialization, finalization.

- For any GUI application using Qt, there is precisely one QApplication object, no matter whether the application has 0, 1, 2 or more windows at any given time. For non-widget based Qt applications, use QCoreApplication instead, as it does not depend on the widgets library.

- Widget : visual element (object)

# Compiling Hello QT Program

- Put your code in a directory (there should be no other non-project related files/codes there)

```
>qmake –project
>qmake HELLO.pro
>make
```

<———— **This command will create a Makefile automatically**

# BUTTON Example

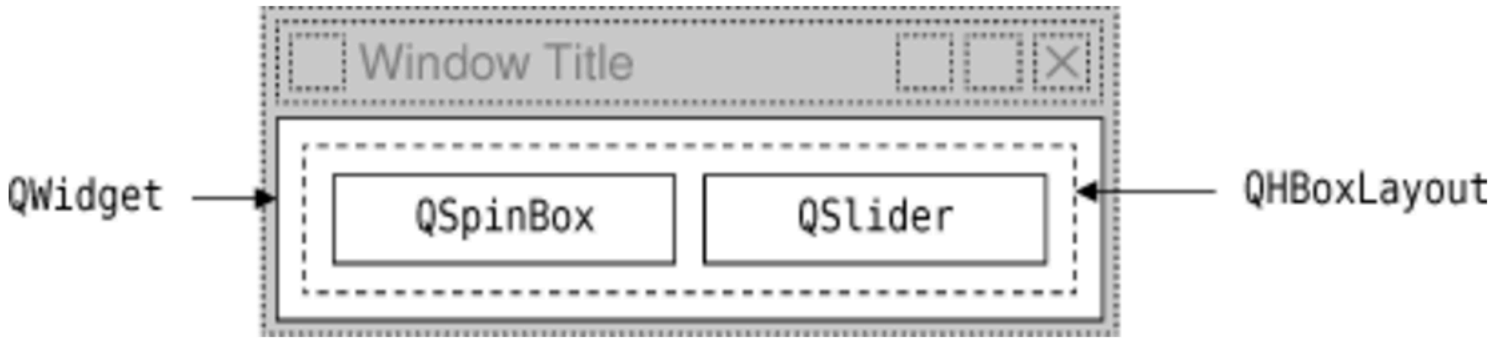

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QPushButton *button = new QPushButton("Quit");
    QObject::connect(button, SIGNAL(clicked()),
                     &app, SLOT(quit()));
    button->show();

    return app.exec();
}
```

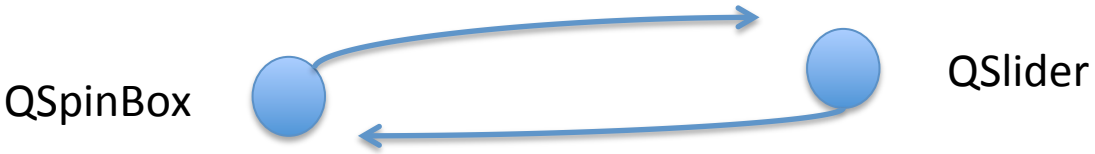
# Signals and Slots

The connect() statement:

**connect(sender, SIGNAL(signal), receiver, SLOT(slot));**

# SLIDER Example




# SLIDER Example Code

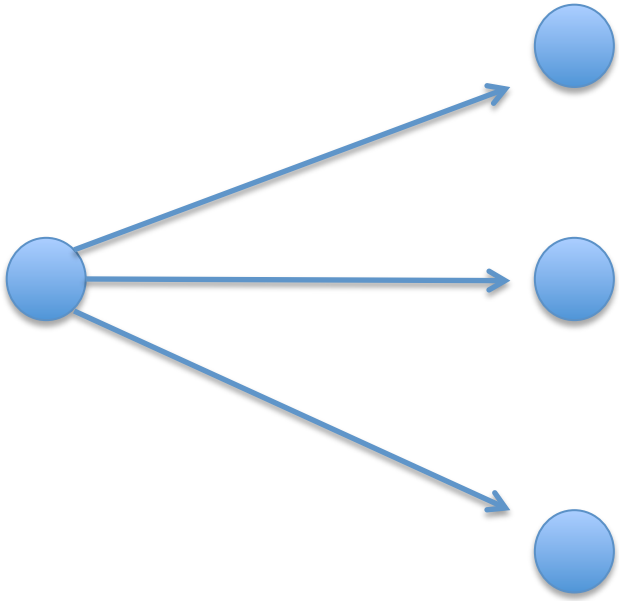
```
#include <QApplication>
#include <QHBoxLayout>
#include <QSlider>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QWidget *window = new QWidget;
    window->setWindowTitle("Enter Your Age");

    QSpinBox *spinBox = new QSpinBox;
    QSlider *slider = new QSlider(Qt::Horizontal);
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);

    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    spinBox->setValue(35);

    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);

    window->show();

    return app.exec();
}
```

# Signals and Slots

- **One signal can be connected to many slots:**

```
connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
        this, SLOT(updateStatusBarIndicator(int)));
```

When the signal is emitted, the slots are called one after the other, in an unspecified order.

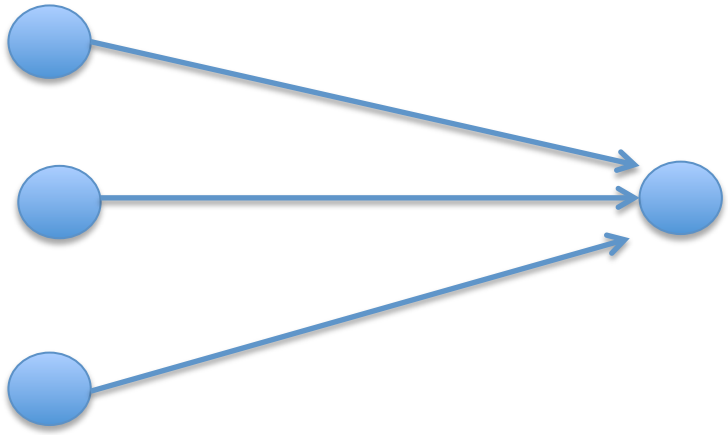


# Signals and Slots

- **Many signals can be connected to the same slot:**

```
connect(lcd, SIGNAL(overflow()),
        this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()),
        this, SLOT(handleMathError()));
```

When either signal is emitted, the slot is called.

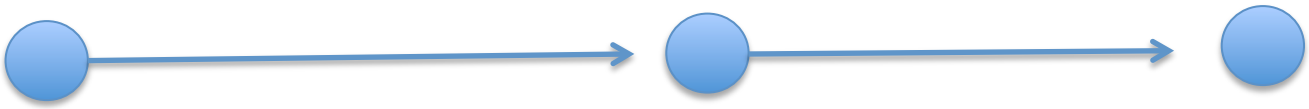


# Signals and Slots

- **A signal can be connected to another signal:**

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
        this, SIGNAL(updateRecord(const QString &)));
```

When the first signal is emitted, the second signal is emitted as well. Apart from that, signal–signal connections are indistinguishable from signal–slot connections.



# Signals and Slots

- **Connections can be removed:**

```
disconnect(lcd, SIGNAL(overflow()),
           this, SLOT(handleMathError()));
```

This is rarely needed, because Qt automatically removes all connection involving an object when that object is deleted.

# Inheritance Tree for the QT Classes seen so far (partial)